Repetición indexada

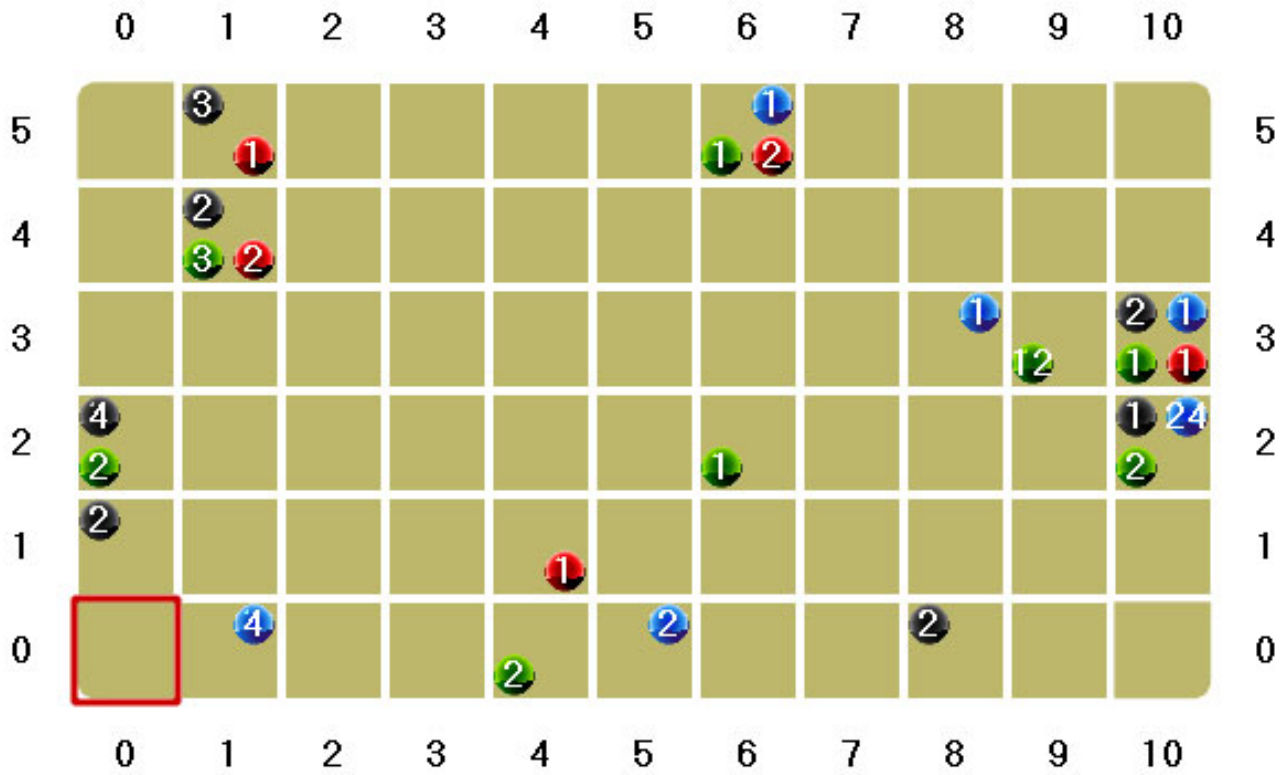
Introducción a la programación

Tecnicatura en programación informática,
Licenciatura en Desarrollo de Software,
Universidad Nacional de Quilmes

Lo que veremos hoy

Repaso (Elementos de Gobstones)

- Tablero: cuadrícula de celdas, que contienen bolitas.
- Cabezal: realiza acciones sobre las celdas del tablero.



Repaso (Programas y comandos)

- Programa Gobstones: texto que describe cómo transformar un tablero
  - program, comandos, propósito, efecto, documentación, tabulación

```
1 /* Proposito: este programa no hace nada */
2 program {
3   Poner(Rojo)
4   Sacar(Rojo)
5 }
```

Repaso (Programas y comandos)

- Programa Gobstones: texto que describe cómo transformar un tablero
  - program, comandos, propósito, efecto, documentación, tabulación

```
1  /* Proposito: este programa no hace nada */
2  program {
3    Poner(Rojo)
4    Sacar(Rojo)
5  }
```

- Comandos:
  - `Poner(<color>)`
  - `Sacar(<color>)`
  - `Mover(<direccion>)`
- Colores:
  - Azul
  - Negro
  - Rojo
  - Verde
- Direcciones:
  - `Norte` ↑
  - `Este` →
  - `Sur` ↓
  - `Oeste` ←

Repaso (Procedimientos simples)

- Procedimientos simples: dan nombre a una secuencia de comandos
  - `procedure` *<NombreMayuscula>() <bloque>*

```
1  /* Proposito: Pone un cuadrado de 2x2 con bolitas de cada color
2   * Cabezal: Queda posicionado en la celda actual
3   */
4  procedure CuadradoMulticolor() {
5    Mover(Este);  PonerUnaDeCada()
6    Mover(Norte); PonerUnaDeCada()
7    Mover(Oeste); PonerUnaDeCada()
8    Mover(Sur);   PonerUnaDeCada()
9  }
```

Repaso (Procedimientos simples)

- Procedimientos simples: dan nombre a una secuencia de comandos
  - `procedure` *<NombreMayuscula>() <bloque>*

```
1  /* Proposito: Pone un cuadrado de 2x2 con bolitas de cada color
2   * Cabezal: Queda posicionado en la celda actual
3   */
4  procedure CuadradoMulticolor() {
5    Mover(Este);   PonerUnaDeCada()
6    Mover(Norte);  PonerUnaDeCada()
7    Mover(Oeste);  PonerUnaDeCada()
8    Mover(Sur);    PonerUnaDeCada()
9  }
```

- Abstracción: ¿Qué hace `PonerUnaDeCada`?

Repaso (Procedimientos simples)

- Procedimientos simples: dan nombre a una secuencia de comandos
  - `procedure` *<NombreMayuscula>() <bloque>*

```
1  /* Proposito: Pone un cuadrado de 2x2 con bolitas de cada color
2   * Cabezal: Queda posicionado en la celda actual
3   */
4  procedure CuadradoMulticolor() {
5    Mover(Este); PonerUnaDeCada()
6    Mover(Norte); PonerUnaDeCada()
7    Mover(Oeste); PonerUnaDeCada()
8    Mover(Sur); PonerUnaDeCada()
9  }
```

- Abstracción: ¿Qué hace `PonerUnaDeCada`?
- Reutilización: ¿Cuántas veces hay que escribir el código para `PonerUnaDeCada`? $\Rightarrow$ Bibliotecas

Repaso (Procedimientos simples)

- Procedimientos simples: dan nombre a una secuencia de comandos
  - `procedure` *<NombreMayuscula>() <bloque>*

```
1  /* Proposito: Pone un cuadrado de 2x2 con bolitas de cada color
2   * Cabezal: Queda posicionado en la celda actual
3   */
4  procedure CuadradoMulticolor() {
5    Mover(Este); PonerUnaDeCada()
6    Mover(Norte); PonerUnaDeCada()
7    Mover(Oeste); PonerUnaDeCada()
8    Mover(Sur); PonerUnaDeCada()
9  }
```

- Abstracción: ¿Qué hace `PonerUnaDeCada`?
- Reutilización: ¿Cuántas veces hay que escribir el código para `PonerUnaDeCada`? $\Rightarrow$ Bibliotecas
- Intercambio: se puede compartir procedimientos (documentación!)

Repaso (División en subtareas)

- Concepto central en la programación
  - Permite reutilización y escalabilidad

Repaso (División en subtareas)

- Concepto central en la programación
  - Permite reutilización y escalabilidad
- Problema: difícil de aprender
  - Dividir en subproblemas sin importar cómo se resuelven los subproblemas
  - Recordar qué cosas ya hicimos y tratar de reutilizarlas
  - Pensar en unidades de procesamiento que tengan sentido

Repaso (División en subtareas)

- Concepto central en la programación
  - Permite reutilización y escalabilidad
- Problema: difícil de aprender
  - Dividir en subproblemas sin importar cómo se resuelven los subproblemas
  - Recordar qué cosas ya hicimos y tratar de reutilizarlas
  - Pensar en unidades de procesamiento que tengan sentido
- Regla del copiar&pegar $\rightarrow$ factorización
  - Si se repite un comportamiento, se extrae

Repaso (Especificación, contratos)

- Problema: intercambio de código para usar como caja negra

Repaso (Especificación, contratos)

- Problema: intercambio de código para usar como caja negra
- Solución: problemas + especificación
  - Propósito: qué problema resuelve un procedimiento.
  - Precondición: condiciones necesarias para garantizar que el problema puede resolverse
  - Tipos de los parámetros: qué valores puede asumir un parámetro (*)

Repaso (Especificación, contratos)

- Problema: intercambio de código para usar como caja negra
- Solución: problemas + especificación
  - Propósito: qué problema resuelve un procedimiento.
  - Precondición: condiciones necesarias para garantizar que el problema puede resolverse
  - Tipos de los parámetros: qué valores puede asumir un parámetro (*)
- Contrato: si se cumple la precondición y los tipos, entonces el procedimiento funciona correctamente

Repaso (Especificación, contratos)

- Problema: intercambio de código para usar como caja negra
- Solución: problemas + especificación
  - Propósito: qué problema resuelve un procedimiento.
  - Precondición: condiciones necesarias para garantizar que el problema puede resolverse
  - Tipos de los parámetros: qué valores puede asumir un parámetro (*)
- Contrato: si se cumple la precondición y los tipos, entonces el procedimiento funciona correctamente
  - Si no se cumple la precondición, el procedimiento puede hacer lo que quiera (incluso BOOM!)

Repaso (Especificación, contratos)

- Problema: intercambio de código para usar como caja negra
- Solución: problemas + especificación
  - Propósito: qué problema resuelve un procedimiento.
  - Precondición: condiciones necesarias para garantizar que el problema puede resolverse
  - Tipos de los parámetros: qué valores puede asumir un parámetro (*)
- Contrato: si se cumple la precondición y los tipos, entonces el procedimiento funciona correctamente
  - Si no se cumple la precondición, el procedimiento puede hacer lo que quiera (incluso BOOM!)
- Permite la construcción de bibliotecas

Repaso (Precondiciones y equivalencia)

- Qué problemas resuelven los siguientes procedimientos
  - Pensar en proposito + precondición.
- ¿Son equivalentes?

```
1 procedure HacerNada() {
2   Poner(Azul)
3   Sacar(Azul)
4 }
```

```
1 procedure HacerNadaBis() {
2   Sacar(Azul)
3   Poner(Azul)
4 }
```

```
1 procedure HacerNadaTri() {
2   Mover(Este)
3   Mover(Oeste)
4 }
```

Repaso (Precondiciones y equivalencia)

- Qué problemas resuelven los siguientes procedimientos
  - Pensar en proposito + precondición.
- ¿Son equivalentes?

```
1 procedure HacerNada() {
2   Poner(Azul)
3   Sacar(Azul)
4 }
```

```
1 procedure HacerNadaBis() {
2   Sacar(Azul)
3   Poner(Azul)
4 }
```

```
1 procedure HacerNadaTri() {
2   Mover(Este)
3   Mover(Oeste)
4 }
```

- Precondición: condiciones que tienen que satisfacer el cabezal, el tablero, y los parámetros para que el problema tenga solución (*).

Repaso (Precondiciones y equivalencia)

- Qué problemas resuelven los siguientes procedimientos
  - Pensar en proposito + precondición.
- ¿Son equivalentes?

```
1 procedure HacerNada() {
2   Poner(Azul)
3   Sacar(Azul)
4 }
```

```
1 procedure HacerNadaBis() {
2   Sacar(Azul)
3   Poner(Azul)
4 }
```

```
1 procedure HacerNadaTri() {
2   Mover(Este)
3   Mover(Oeste)
4 }
```

- Precondición: condiciones que tienen que satisfacer el cabezal, el tablero, y los parámetros para que el problema tenga solución (*).
- Procedimientos equivalentes: mismo propósito, precondiciones y tipos.

Repaso (Parámetros)

- Formalmente: identificador que permite denotar distintos valores cuando se invoca un procedimiento
- Intuitivamente: cajitas, asteriscos, etc, que reemplazan un valor fijo por una expresión que denota un valor cualquiera
- Motivación estudio: recordar la regla del copy&paste

```
1  /* Pone una bolita roja
2   * en la celda actual y
3   * mueve el cabezal al este.
4   * Prec: el cabezal no se
5   * encuentra en la ultima col
6   */
7  procedure PonerRojaYMoverAlEste() {
8    Poner(Rojo)
9    Mover(Este)
10 }
```

```
1  /* Pone una bolita azul
2   * en la celda actual y
3   * mueve el cabezal al oeste.
4   * Prec: el cabezal no se
5   * encuentra en la primer col
6   */
7  procedure PonerAzulYMoverAlOeste() {
8    Poner(Azul)
9    Mover(Oeste)
10 }
```

Repaso (Parámetros)

- Formalmente: identificador que permite denotar distintos valores cuando se invoca un procedimiento
- Intuitivamente: cajitas, asteriscos, etc, que reemplazan un valor fijo por una expresión que denota un valor cualquiera
- Motivación estudio: recordar la regla del copy&paste

```
1  /* Pone una bolita ***
2   * en la celda actual y
3   * mueve el cabezal al este.
4   * Prec: el cabezal no se
5   * encuentra en la ultima col
6   */
7  procedure PonerYMoverAlEste(***) {
8    Poner(***)
9    Mover(Este)
10 }
```

```
1  /* Pone una bolita ***
2   * en la celda actual y
3   * mueve el cabezal al oeste.
4   * Prec: el cabezal no se
5   * encuentra en la primer col
6   */
7  procedure PonerYMoverAlOeste(***) {
8    Poner(***)
9    Mover(Oeste)
10 }
```

Repaso (Parámetros)

- Formalmente: identificador que permite denotar distintos valores cuando se invoca un procedimiento
- Intuitivamente: cajitas, asteriscos, etc, que reemplazan un valor fijo por una expresión que denota un valor cualquiera
- Motivación estudio: recordar la regla del copy&paste

```
1  /* Pone una bolita ***
2   * en la celda actual y
3   * mueve el cabezal al este.
4   * Prec: el cabezal no se
5   * encuentra en la ultima col
6   */
7  procedure PonerYMoverAlEste(***) {
8    Poner(***)
9    Mover(Este)
10 }
```

```
1  /* Pone una bolita ***
2   * en la celda actual y
3   * mueve el cabezal al oeste.
4   * Prec: el cabezal no se
5   * encuentra en la primer col
6   */
7  procedure PonerYMoverAlOeste(***) {
8    Poner(***)
9    Mover(Oeste)
10 }
```

Repaso (Parámetros)

- Formalmente: identificador que permite denotar distintos valores cuando se invoca un procedimiento
- Intuitivamente: cajitas, asteriscos, etc, que reemplazan un valor fijo por una expresión que denota un valor cualquiera
- Motivación estudio: recordar la regla del copy&paste

```
1  /* Pone una bolita ***
2   * en la celda actual y
3   * mueve el cabezal al ###.
4   * Prec: el cabezal no se
5   * encuentra en la ultima ### col
6   */
7  procedure PonerYMover(***, ###) {
8    Poner(***)
9    Mover(###)
10 }
```

```
1  /* Pone una bolita ***
2   * en la celda actual y
3   * mueve el cabezal al ###.
4   * Prec: el cabezal no se
5   * encuentra en la primer ### col
6   */
7  procedure PonerYMover(***, ###) {
8    Poner(***)
9    Mover(###)
10 }
```

Repaso (Parámetros)

- Formalmente: identificador que permite denotar distintos valores cuando se invoca un procedimiento
- Intuitivamente: cajitas, asteriscos, etc, que reemplazan un valor fijo por una expresión que denota un valor cualquiera
- Motivación estudio: recordar la regla del copy&paste

```
1  /* Pone una bolita de color c
2   * en la celda actual y
3   * mueve el cabezal en direccion d.
4   * Precondicion: el cabezal no se
5   * encuentra en la ultima columna en direccion d
6   */
7  procedure PonerYMover(c, d) {
8    Poner(c)
9    Mover(d)
10 }
```

Repaso (Invocación)

- Para invocar un procedimiento dentro de otro, escribimos `NombreProc(argumentos)`

Repaso (Invocación)

- Para invocar un procedimiento dentro de otro, escribimos `NombreProc(argumentos)`
- Cada argumento denota un único valor

Repaso (Invocación)

- Para invocar un procedimiento dentro de otro, escribimos `NombreProc(argumentos)`
- Cada argumento denota un único valor
- La lista de argumentos se separa por comas

Repaso (Invocación)

- Para invocar un procedimiento dentro de otro, escribimos `NombreProc(argumentos)`
- Cada argumento denota un único valor
- La lista de argumentos se separa por comas
- El primer argumento corresponde al primer parámetro, el segundo argumento al segundo parámetro, etc.
  - `PonerYMover(Azul, Este)`, `PonerYMover(Verde, Oeste)`

Repaso (Invocación)

- Para invocar un procedimiento dentro de otro, escribimos `NombreProc(argumentos)`
- Cada argumento denota un único valor
- La lista de argumentos se separa por comas
- El primer argumento corresponde al primer parámetro, el segundo argumento al segundo parámetro, etc.
  - `PonerYMover(Azul, Este)`, `PonerYMover(Verde, Oeste)`
- Funcionamiento: se interrumpe la ejecución del procedimiento actual y se ejecuta el procedimiento invocado.
  - Cada paramétro pasa a denotar el mismo valor que el argumento correspondiente.
  - Cuando el procedimiento invocado termina, se continúa ejecutando el procedimiento llamador.

Repaso (Invocación)

- Para invocar un procedimiento dentro de otro, escribimos `NombreProc(argumentos)`
- Cada argumento denota un único valor
- La lista de argumentos se separa por comas
- El primer argumento corresponde al primer parámetro, el segundo argumento al segundo parámetro, etc.
  - `PonerYMover(Azul, Este)`, `PonerYMover(Verde, Oeste)`
- Funcionamiento: se interrumpe la ejecución del procedimiento actual y se ejecuta el procedimiento invocado.
  - Cada paramétro pasa a denotar el mismo valor que el argumento correspondiente.
  - Cuando el procedimiento invocado termina, se continúa ejecutando el procedimiento llamador.

- Contrato: si los argumentos y el tablero satisfacen la precondición, entonces el procedimiento cumple su propósito.

Repaso (alcance de parámetros)

- ¿Qué relación hay entre el parámetro c de PonerDos y PonerCuatro?

```
1  // Pone dos bolita de color c sin mover el cabezal
2  procedure PonerDos(c) {
3    Poner(c)
4    Poner(c)
5  }
6
7
8  // Pone cuatro bolitas de color c sin mover el cabezal
9  procedure PonerCuatro(c) {
10   PonerDos(c)
11   PonerDos(c)
12 }
```

Repaso (alcance de parámetros)

- ¿Qué relación hay entre el parámetro c de PonerDos y PonerCuatro?

```
1  // Pone dos bolita de color c sin mover el cabezal
2  procedure PonerDos(c) {
3    Poner(c)
4    Poner(c)
5  }
6
7
8  // Pone cuatro bolitas de color c sin mover el cabezal
9  procedure PonerCuatro(c) {
10   PonerDos(c)
11   PonerDos(c)
12 }
```

- Alcance de un parámetro: bloque en el que es válido
- Nombre y apellido de un parámetro
  - "c de PonerDos" vs. "c de PonerCuatro"

Repetición simple: motivación

- ¿Cómo hacemos para poner 4 bolitas rojas?

Repetición simple: motivación

- ¿Cómo hacemos para poner 4 bolitas rojas?
- ¿Cómo hacemos para poner 10 bolitas rojas?

Repetición simple: motivación

- ¿Cómo hacemos para poner 4 bolitas rojas?
- ¿Cómo hacemos para poner 10 bolitas rojas?
- ¿Cómo hacemos para poner 10000 bolitas rojas?

Repetición simple: motivación

- ¿Cómo hacemos para poner 4 bolitas rojas?
- ¿Cómo hacemos para poner 10 bolitas rojas?
- ¿Cómo hacemos para poner 10000 bolitas rojas?

```
1  // Proposito: Agrega 10000 bolitas rojas
2  procedure PonerDiezMilRojas() {
3    repeat(10000) {
4      Poner(Rojo)
5    }
6  }
```

Repetición simple: forma general

Repetición simple (`repeat`)

`repeat(`*<numero>*`)`
<bloque a repetir>

donde,

- *<numero>* denota un número n cualquiera

- El `repeat` se puede escribir en los mismos lugares que los comandos
  - Bloque: secuencia de comandos y repeticiones (*)
- Funcionamiento: se ejecuta `bloque a repetir` tantas veces como el número n. Si $n \leq 0$, entonces no hace nada.

Repeat y parámetros

- Hacer un procedimiento `PonerVeinte(c)` que ponga veinte bolitas de color `c` en la celda actual.

Repeat y parámetros

- Hacer un procedimiento `PonerVeinte(c)` que ponga veinte bolitas de color `c` en la celda actual.
- Hacer un procedimiento `PonerTreinta(c)` que ponga treinta bolitas de color `c` en la celda actual.

Repeat y parámetros

- Hacer un procedimiento `PonerVeinte(c)` que ponga veinte bolitas de color `c` en la celda actual.
- Hacer un procedimiento `PonerTreinta(c)` que ponga treinta bolitas de color `c` en la celda actual.
- Hacer un procedimiento `PonerMil(c)` que ponga 1000 bolitas de color `c` en la celda actual.

Repeat y parámetros

- Hacer un procedimiento `PonerVeinte(c)` que ponga veinte bolitas de color `c` en la celda actual.
- Hacer un procedimiento `PonerTreinta(c)` que ponga treinta bolitas de color `c` en la celda actual.
- Hacer un procedimiento `PonerMil(c)` que ponga 1000 bolitas de color `c` en la celda actual.

- Hacer un procedimiento `PonerN(c, n)` que ponga `n` bolitas de color `c` en la celda actual.

Repeat y parámetros

- Hacer un procedimiento `PonerVeinte(c)` que ponga veinte bolitas de color `c` en la celda actual.
- Hacer un procedimiento `PonerTreinta(c)` que ponga treinta bolitas de color `c` en la celda actual.
- Hacer un procedimiento `PonerMil(c)` que ponga 1000 bolitas de color `c` en la celda actual.

- Hacer un procedimiento `PonerN(c, n)` que ponga `n` bolitas de color `c` en la celda actual.
- ¿Qué hace `PonerN(c, -10)`?

Repeat y parámetros

- Hacer un procedimiento `PonerVeinte(c)` que ponga veinte bolitas de color `c` en la celda actual.
- Hacer un procedimiento `PonerTreinta(c)` que ponga treinta bolitas de color `c` en la celda actual.
- Hacer un procedimiento `PonerMil(c)` que ponga 1000 bolitas de color `c` en la celda actual.

- Hacer un procedimiento `PonerN(c, n)` que ponga `n` bolitas de color `c` en la celda actual.
- ¿Qué hace `PonerN(c, -10)`?

- Cualquier parámetro (u expresión) que denote un número puede usarse como cantidad de una repetición simple.

Formalizando I: valores y expresiones

- Valores: colores, direcciones, números.

Formalizando I: valores y expresiones

- Valores: colores, direcciones, números.
- Expresión: cadena de símbolos (i.e., palabras) que denota un valor (*)

Formalizando I: valores y expresiones

- Valores: colores, direcciones, números.
- Expresión: cadena de símbolos (i.e., palabras) que denota un valor (*)
- Expresiones literales (o atómicas): denotan directamente un valor
  - Colores: `Azul`, `Negro`, `Rojo`, `Verde`
  - Direcciones: `Norte`, `Este`, `Sur`, `Oeste`
  - Números (enteros): ..., `-3`, `-2`, `-1`, `0`, `1`, `2`, `3`, ...

Formalizando I: valores y expresiones

- Valores: colores, direcciones, números.
- Expresión: cadena de símbolos (i.e., palabras) que denota un valor (*)
- Expresiones literales (o atómicas): denotan directamente un valor
  - Colores: `Azul`, `Negro`, `Rojo`, `Verde`
  - Direcciones: `Norte`, `Este`, `Sur`, `Oeste`
  - Números (enteros): ..., `-3`, `-2`, `-1`, `0`, `1`, `2`, `3`, ...
- Expresiones no literales: denotan valores indirectamente
  - identificadores: e.g. parámetros

Formalizando I: valores y expresiones

- Valores: colores, direcciones, números.
- Expresión: cadena de símbolos (i.e., palabras) que denota un valor (*)
- Expresiones literales (o atómicas): denotan directamente un valor
  - Colores: `Azul`, `Negro`, `Rojo`, `Verde`
  - Direcciones: `Norte`, `Este`, `Sur`, `Oeste`
  - Números (enteros): ..., `-3`, `-2`, `-1`, `0`, `1`, `2`, `3`, ...
- Expresiones no literales: denotan valores indirectamente
  - identificadores: e.g. parámetros
  - operaciones:
    `1 + 2`, `7 - 8`, `minColor()`, `maxColor()`, `nroBolitas(Azul)`, etc

Formalizando I: valores y expresiones

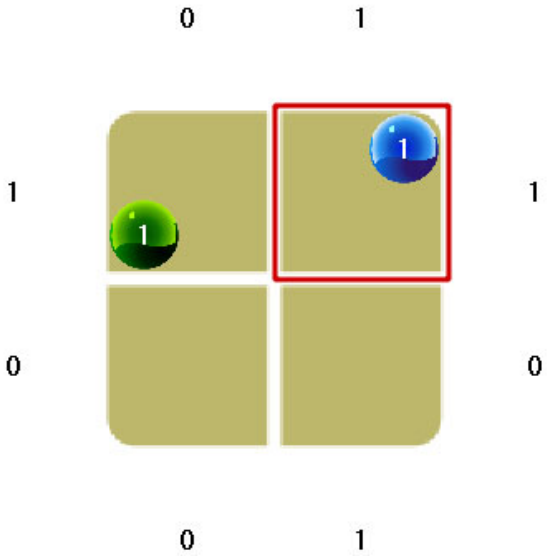
- Valores: colores, direcciones, números.
- Expresión: cadena de símbolos (i.e., palabras) que denota un valor (*)
- Expresiones literales (o atómicas): denotan directamente un valor
  - Colores: `Azul`, `Negro`, `Rojo`, `Verde`
  - Direcciones: `Norte`, `Este`, `Sur`, `Oeste`
  - Números (enteros): ..., `-3`, `-2`, `-1`, `0`, `1`, `2`, `3`, ...
- Expresiones no literales: denotan valores indirectamente
  - identificadores: e.g. parámetros
  - operaciones:
    `1 + 2`, `7 - 8`, `minColor()`, `maxColor()`, `nroBolitas(Azul)`, etc
- Evaluar: determinar qué valor denota una expresión

Formalizando I: valores y expresiones

- Valores: colores, direcciones, números.
- Expresión: cadena de símbolos (i.e., palabras) que denota un valor (*)
- Expresiones literales (o atómicas): denotan directamente un valor
  - Colores: `Azul`, `Negro`, `Rojo`, `Verde`
  - Direcciones: `Norte`, `Este`, `Sur`, `Oeste`
  - Números (enteros): ..., `-3`, `-2`, `-1`, `0`, `1`, `2`, `3`, ...
- Expresiones no literales: denotan valores indirectamente
  - identificadores: e.g. parámetros
  - operaciones:
    `1 + 2`, `7 - 8`, `minColor()`, `maxColor()`, `nroBolitas(Azul)`, etc
- Evaluar: determinar qué valor denota una expresión
- Cualquier expresión se puede en lugar de un literal

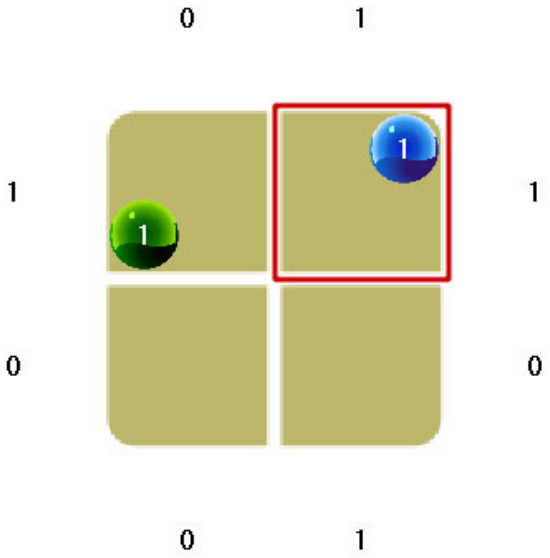
Formalizando II: expresiones y operaciones

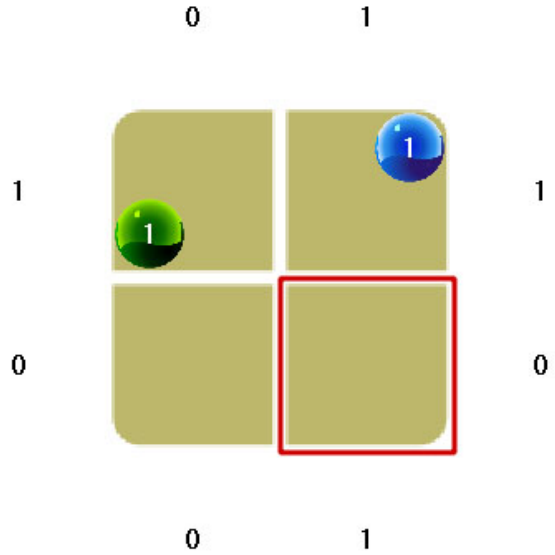
- Distintas expresiones pueden denotar el mismo valor
  - e.g. `1`, `5 - 4`, `-2+ 3`, `nroBolitas(Azul)`



Formalizando II: expresiones y operaciones

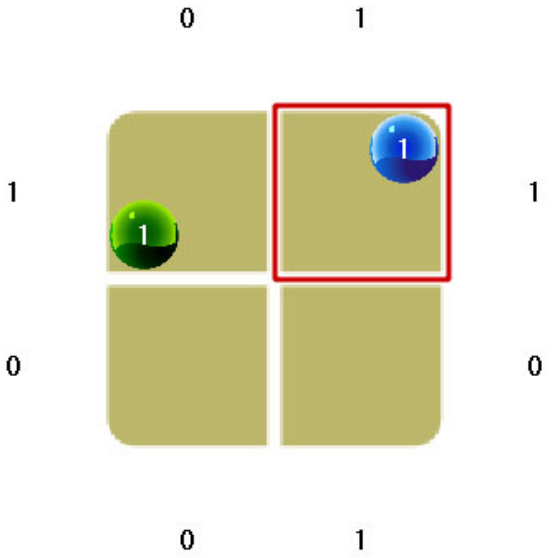
- Distintas expresiones pueden denotar el mismo valor
  - e.g. `1`, `5 - 4`, `-2+ 3`, `nroBolitas(Azul)`
- La misma expresión puede denotar valores distintos en instantes diferentes
  - `nroBolitas(Azul)`

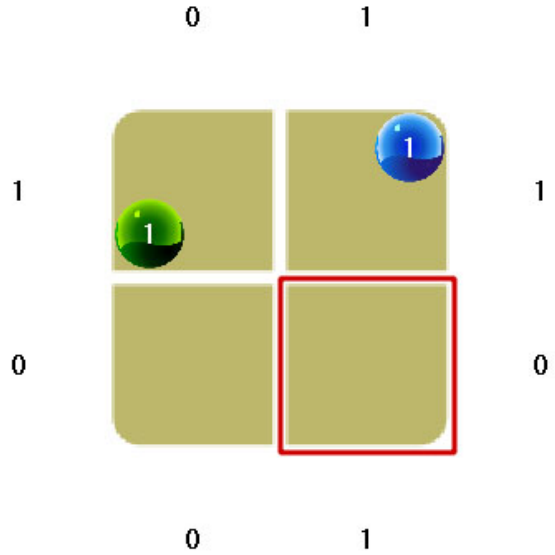




Formalizando II: expresiones y operaciones

- Distintas expresiones pueden denotar el mismo valor
  - e.g. `1`, `5 - 4`, `-2+ 3`, `nroBolitas(Azul)`
- La misma expresión puede denotar valores distintos en instantes diferentes
  - `nroBolitas(Azul)`



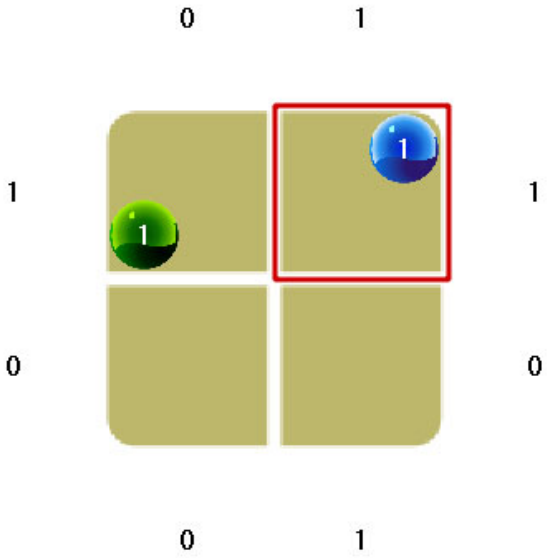


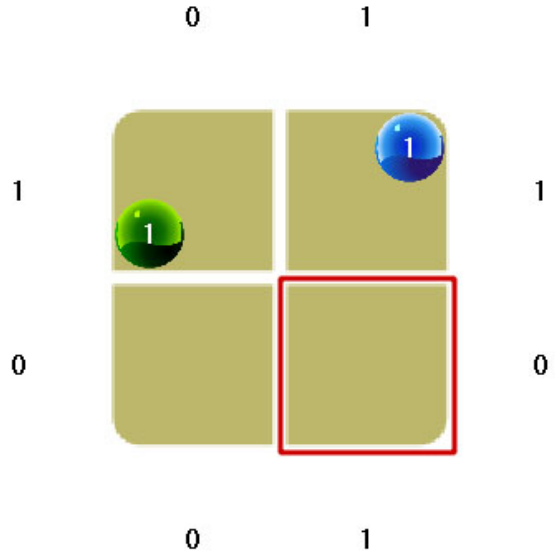
Operaciones

- denotan valores a partir del estado del tablero y los parámetros

Formalizando II: expresiones y operaciones

- Distintas expresiones pueden denotar el mismo valor
  - e.g. `1`, `5 - 4`, `-2+ 3`, `nroBolitas(Azul)`
- La misma expresión puede denotar valores distintos en instantes diferentes
  - `nroBolitas(Azul)`



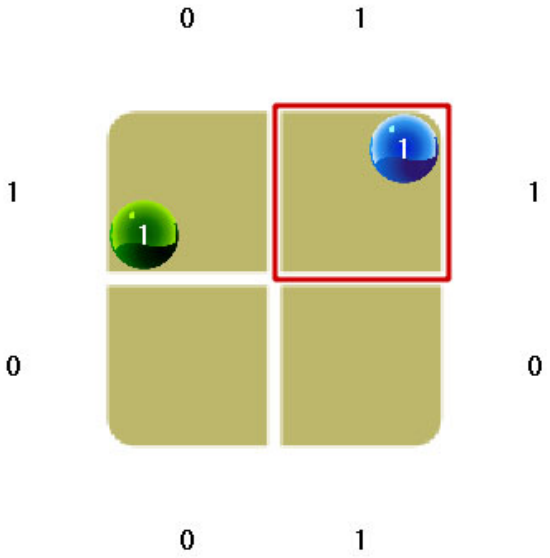


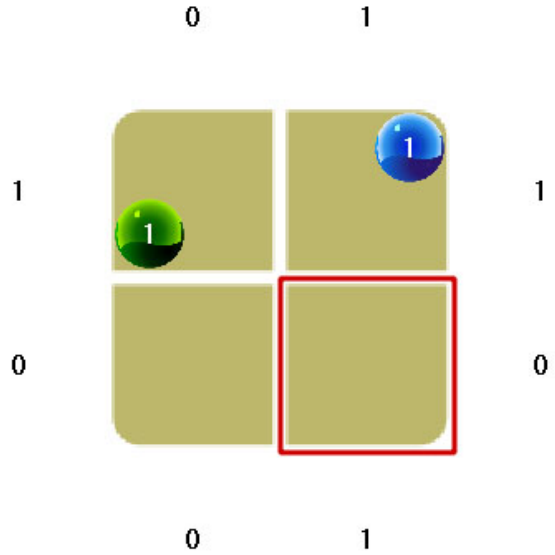
Operaciones

- denotan valores a partir del estado del tablero y los parámetros
- no modifican el estado del tablero

Formalizando II: expresiones y operaciones

- Distintas expresiones pueden denotar el mismo valor
  - e.g. `1`, `5 - 4`, `-2+ 3`, `nroBolitas(Azul)`
- La misma expresión puede denotar valores distintos en instantes diferentes
  - `nroBolitas(Azul)`





Operaciones

- denotan valores a partir del estado del tablero y los parámetros
- no modifican el estado del tablero
- se escriben en minúscula y tienen una lista de argumentos

Formalizando II: operaciones

Algunas operaciones con colores: *<c>* color

- `minColor()`: mínimo color en orden alfabético (`Azul`)
- `maxColor()`: máximo color en orden alfabético (`Verde`)
- `siguiente(`*<c>*`)`: color siguiente a *<c>* en orden alfabético
  - `siguiente(maxColor())` denota `minColor()`
- `previo(`*<c>*`)`: color anterior a *<c>* en orden alfabético
  - `previo(minColor())` denota `maxColor()`
- `nroBolitas(`*<c>*`)`: cantidad de bolitas de color *<c>* en la celda actual

Formalizando II: operaciones

Algunas operaciones con direcciones: *<d>* dirección

- `minDir()`: mínima dirección en el sentido del reloj (`Norte`)
- `maxDir()`: máxima dirección en el sentido del reloj (`Oeste`)
- `siguiente(`*<d>*`)`: dirección siguiente a *<d>* en sentido del reloj.
  - `siguiente(maxDir())` denota `minDir()`
- `previo(`*<d>*`)`: dirección anterior a *<d>* en sentido del reloj
  - `previo(minDir())` denota `maxDir()`
- `opuesto(`*<d>*`)`: dirección cardinal opuesta a *<d>*

Formalizando II: operaciones

Algunas operaciones con números: *<n>* y *<m>* números

- *<n>* `+` *<m>*: denota la suma de *<n>* y *<m>*
- *<n>* `-` *<m>*: denota la resta de *<n>* y *<m>*
- *<n>* `*` *<m>*: denota la multiplicación *<n>* y *<m>*
- *<n>* `div` *<m>*: denota la división entera *<n>* y *<m>*
- *<n>* `mod` *<m>*: denota el resto de la división entera de *<n>* y *<m>*
- *<n>* `^` *<m>*: denota el resultado de elevar *<n>* por *<m>*, i.e., n^m
- `-` *<n>*: denota `0 -` *<n>*

Formalizando II: operaciones

- ¿Qué hace el siguiente procedimiento?

```
1  procedure AdivinarComportamiento() {
2    Poner(minColor())
3    Mover(siguiente(minDir()))
4    Poner(siguiente(minColor()))
5    Mover(minDir())
6    Poner(previo(maxColor()))
7    Mover(previo(minDir()))
8    Poner(maxColor())
9    Mover(opuesto(minDir()))
10 }
```

Formalizando II: operaciones

- ¿Qué hace el siguiente procedimiento?

```
1  procedure AdivinarComportamiento() {
2    Poner(minColor())
3    Mover(siguiente(minDir()))
4    Poner(siguiente(minColor()))
5    Mover(minDir())
6    Poner(previo(maxColor()))
7    Mover(previo(minDir()))
8    Poner(maxColor())
9    Mover(opuesto(minDir()))
10 }
```

- Recordar: cada vez que se requiera un valor se puede usar una expresión
  - Los literales son sólo una forma de expresión

Más ejemplos motivadores

- ¿Qué hace el procedimiento `MoverN(Este, 5)`?
- ¿Cuál es el propósito de `MoverN`?
- ¿Cuál es su precondición?

```
1  procedure MoverN(d, n) {
2    repeat(|n|) {
3      Mover(d)
4    }
5  }
```

Más ejemplos motivadores

- ¿Qué hace el procedimiento `MoverN(Este, 5)`?
- ¿Cuál es el propósito de `MoverN`?
- ¿Cuál es su precondición?

```
1  procedure MoverN(d, n) {
2    repeat(|n|) {
3      Mover(d)
4    }
5  }
```

- Recordar: para describir la cantidad podemos usar expresiones!

Ejemplo de expresiones

- ¿Qué hace el procedimiento `Duplicar(Azul)`?
- ¿Cuál es el propósito de `Duplicar`?
- ¿Cuál es su precondición?

```
1  procedure Duplicar(c) {
2    PonerN(c, nroBolitas(c))
3  }
```

Ejemplo de expresiones

- ¿Qué hace el procedimiento `Duplicar(Azul)`?
- ¿Cuál es el propósito de `Duplicar`?
- ¿Cuál es su precondición?

```
1  procedure Duplicar(c) {
2    PonerN(c, nroBolitas(c))
3  }
```

- Recordar: cualquier expresión puede usarse como argumento
- Ejercicio: escribir la función `SacarTodas` invocando a `SacarN`

Más ejemplos motivadores

- ¿Qué hace el procedimiento `RepartirBolitas(Este, Azul)`?
- ¿Cuál es el propósito de `RepartirBolitas` (cabezal!)?
- ¿Cuál es su precondición?

```
1  procedure RepartirBolitas(d, c) {
2    repeat(nroBolitas(c)) {
3      Mover(d)
4      Poner(c)
5    }
6  }
```

Más ejemplos motivadores

- ¿Qué hace el procedimiento `RepartirBolitas(Este, Azul)`?
- ¿Cuál es el propósito de `RepartirBolitas` (cabezal!)?
- ¿Cuál es su precondición?

```
1  procedure RepartirBolitas(d, c) {
2    repeat(nroBolitas(c)) {
3      Mover(d)
4      Poner(c)
5    }
6  }
```

- Ejercicio: ¿cómo hacemos para dejar el cabezal en la celda inicial?

Formalizando III: argumentos y expresiones

- Formalmente, cada comando u operación tiene una lista de parámetros, que son simplemente identificadores
  - `Poner(<color>), procedure PonerN(c,n) <bloque>`
  - `nroBolitas(<color>), n + m, repeat(<numero>) <bloque>`

Formalizando III: argumentos y expresiones

- Formalmente, cada comando u operación tiene una lista de parámetros, que son simplemente identificadores
  - `Poner(<color>), procedure PonerN(c,n) <bloque>`
  - `nroBolitas(<color>), n + m, repeat(<numero>) <bloque>`
- Al invocar un comando u operación, se debe incluir una lista con tantos argumentos como parámetros tenga el comando u operación invocado
  - Bien: `PonerN(Azul, 3), nroBolitas(Rojo), repeat (4) {}`.
  - Mal: `PonerN(Azul), nroBolitas(Rojo, Azul), repeat() {}`

Formalizando III: argumentos y expresiones

- Formalmente, cada comando u operación tiene una lista de parámetros, que son simplemente identificadores
  - `Poner(<color>), procedure PonerN(c,n) <bloque>`
  - `nroBolitas(<color>), n + m, repeat(<numero>) <bloque>`
- Al invocar un comando u operación, se debe incluir una lista con tantos argumentos como parámetros tenga el comando u operación invocado
  - Bien: `PonerN(Azul, 3), nroBolitas(Rojo), repeat (4) {}`.
  - Mal: `PonerN(Azul), nroBolitas(Rojo, Azul), repeat() {}`
- Cada argumento debe ser una expresión que puede no ser literal

Formalizando III: argumentos y expresiones

- Formalmente, cada comando u operación tiene una lista de parámetros, que son simplemente identificadores
  - `Poner(<color>)`, `procedure PonerN(c,n) <bloque>`
  - `nroBolitas(<color>)`, `n + m`, `repeat(<numero>) <bloque>`
- Al invocar un comando u operación, se debe incluir una lista con tantos argumentos como parámetros tenga el comando u operación invocado
  - Bien: `PonerN(Azul, 3)`, `nroBolitas(Rojo)`, `repeat (4) {}`.
  - Mal: `PonerN(Azul)`, `nroBolitas(Rojo, Azul)`, `repeat() {}`
- Cada argumento debe ser una expresión que puede no ser literal
- La unica restricción es que el tipado sea correcto; clase que viene
  - Bien:
    `siguiente(Rojo), PonerN(Azul, nroBolitas(Verde) + 2)`
  - Bien: `repeat (nroBolitas(Azul) div 12)`
  - Mal: `nroBolitas(Poner(Rojo)), Poner(Poner(Verde))`

Formalizando III: argumentos y expresiones

- Formalmente, cada comando u operación tiene una lista de parámetros, que son simplemente identificadores
  - `Poner(<color>)`, `procedure PonerN(c,n) <bloque>`
  - `nroBolitas(<color>)`, `n + m`, `repeat(<numero>) <bloque>`
- Al invocar un comando u operación, se debe incluir una lista con tantos argumentos como parámetros tenga el comando u operación invocado
  - Bien: `PonerN(Azul, 3)`, `nroBolitas(Rojo)`, `repeat (4) {}`.
  - Mal: `PonerN(Azul)`, `nroBolitas(Rojo, Azul)`, `repeat() {}`
- Cada argumento debe ser una expresión que puede no ser literal
- La unica restricción es que el tipado sea correcto; clase que viene
  - Bien:
    `siguiente(Rojo), PonerN(Azul, nroBolitas(Verde) + 2)`
  - Bien: `repeat (nroBolitas(Azul) div 12)`
  - Mal: `nroBolitas(Poner(Rojo)), Poner(Poner(Verde))`
- Recíprocamente, las expresiones sólo pueden usarse como argumentos.
  - Mal: `repeat(1){ Azul }, repeat(1){ nroBolitas(Azul) }`

Formalizando III: parámetros y expresiones

- Dentro de un procedimiento, cada parametro es una expresión
  - Puede usarse como argumento y sólo como argumento
  - Denota un valor desconocido hasta que el procedimiento se ejecuta.

```
1 //c es un parametro y se usa como cualquier expresion
2 procedure Duplicar(c) {
3   PonerN(c, nroBolitas(c))
4 }
```

- Cuando el procedimiento es invocado, el parámetro denota el valor correspondiente a su argumento durante esa invocación.
  - Durante la invocación de `Duplicar(Azul)`, su parámetro `c` denota el valor `Azul`
  - Cuando se ejecute luego la invocación de `PonerN`, su parámetro `n` denotara la cantidad de bolitas azules de la celda actual.

Repetición indexada: motivación

- ¿Cómo hacemos para poner 1 bolita azul en la celda actual, 2 en la celda siguiente al este, 3 en la celda a dos de distancia al este, y 4 en la celda a distancia 3 al este?

Repetición indexada: motivación

- ¿Cómo hacemos para poner 1 bolita azul en la celda actual, 2 en la celda siguiente al este, 3 en la celda a dos de distancia al este, y 4 en la celda a distancia 3 al este?
- ¿Y si queremos hacerlo 10 veces?

Repetición indexada: motivación

- ¿Cómo hacemos para poner 1 bolita azul en la celda actual, 2 en la celda siguiente al este, 3 en la celda a dos de distancia al este, y 4 en la celda a distancia 3 al este?
- ¿Y si queremos hacerlo 10 veces?
- ¿Y en el caso de que queramos 10000 veces?

Repetición indexada: motivación

- ¿Cómo hacemos para poner 1 bolita azul en la celda actual, 2 en la celda siguiente al este, 3 en la celda a dos de distancia al este, y 4 en la celda a distancia 3 al este?
- ¿Y si queremos hacerlo 10 veces?
- ¿Y en el caso de que queramos 10000 veces?

```
1  /* Proposito: Hace una progresi n de 10000 bolitas azules
2   * Cabezal: a 10000 celdas al este
3   * Precondicion: Hay al menos 10000 celdas al este de la actual
4   */
5  procedure ProgresionDiezMil() {
6    foreach i in [1..10000] {
7      PonerN(Rojo, i)
8      Mover(Este)
9    }
10 }
```

Repetición indexada: forma general

Repetición indexada (`foreach`)

`foreach` *<indice>* `in` *<rango>*
<bloque a repetir>

donde,

- *<indice>* es un identificador
- *<rango>* describe un rango `[n..m]` (a ser definido)

- El `foreach` se puede escribir en los mismos lugares que los comandos
  - Bloque: secuencia de comandos y repeticiones (*)
- Funcionamiento: por cada elemento del rango, se ejecuta *<bloque a repetir>*

Formalizando: rangos

- Informalmente: lo que se escribe al final del `foreach`

Formalizando: rangos

- Informalmente: lo que se escribe al final del `foreach`
- Formalmente: secuencia ordenada (menor a mayor) y finita de valores contiguos

Formalizando: rangos

- Informalmente: lo que se escribe al final del `foreach`
- Formalmente: secuencia ordenada (menor a mayor) y finita de valores contiguos

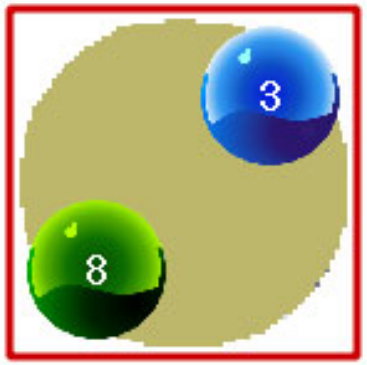
- En GOBSTONES se escribe *<primero>..<ultimo>*
  - *<primero>* y *<ultimo>* son expresiones
  - Si *<ultimo>* es anterior a *<primero>*, entonces el rango es vacío
  - `foreach` ejecuta *<bloque a repetir>* una vez por cada elemento del rango

Ejercitando rangos

- Indicar los valores (en orden) de los siguiente rangos
  - `[1..10]`:
  - `[-3..3]`:
  - `[10..1]`:
  - `[nroBolitas(Azul)..nroBolitas(Verde)]`:

0

0



0

0

Índice: motivación

- Suponga `DibujarLinea(n, d, c)` donde el cabezal queda en la celda actual

Índice: motivación

- Suponga `DibujarLinea(n, d, c)` donde el cabezal queda en la celda actual
- ¿Qué hace el procedimiento `Escalera(5, Azul)`?
- ¿Cuál es el propósito de `Escalera(n, c)`?

```
1  procedure Escalera(n,c) {
2    foreach i in [1..n] {
3      DibujarLinea(i, Este, c)
4      Mover(Norte)
5    }
6  }
```

Índice: motivación

- Suponga `DibujarLinea(n, d, c)` donde el cabezal queda en la celda actual
- ¿Qué hace el procedimiento `Escalera(5, Azul)`?
- ¿Cuál es el propósito de `Escalera(n, c)`?

```
1  procedure Escalera(n,c) {
2    foreach i in [1..n] {
3      DibujarLinea(i, Este, c)
4      Mover(Norte)
5    }
6  }
```

- Los índices son expresiones que denotan los valores del rango
- Como cualquier expresión se pueden usar como argumentos y SOLO como argumentos

Formalizando IV: índices

- `foreach` ejecuta *<bloque a repetir>* una vez por cada elemento del rango

Formalizando IV: índices

- `foreach` ejecuta *<bloque a repetir>* una vez por cada elemento del rango
- Esta ejecución es en orden

Formalizando IV: índices

- `foreach` ejecuta *<bloque a repetir>* una vez por cada elemento del rango
- Esta ejecución es en orden
  - Primer repetición: corresponde al primer elemento del rango

Formalizando IV: índices

- `foreach` ejecuta *<bloque a repetir>* una vez por cada elemento del rango
- Esta ejecución es en orden
  - Primer repetición: corresponde al primer elemento del rango
  - Segunda repetición: corresponde al segundo elemento del rango

Formalizando IV: índices

- `foreach` ejecuta *<bloque a repetir>* una vez por cada elemento del rango
- Esta ejecución es en orden
  - Primer repetición: corresponde al primer elemento del rango
  - Segunda repetición: corresponde al segundo elemento del rango
  - *n*-ésima repetición: corresponde al *n*-ésimo elemento del rango

Formalizando IV: índices

- `foreach` ejecuta *<bloque a repetir>* una vez por cada elemento del rango
- Esta ejecución es en orden
  - Primer repetición: corresponde al primer elemento del rango
  - Segunda repetición: corresponde al segundo elemento del rango
  - *n*-ésima repetición: corresponde al *n*-ésimo elemento del rango
  - Última repetición: corresponde al último elemento del rango

Formalizando IV: índices

- `foreach` ejecuta *<bloque a repetir>* una vez por cada elemento del rango
- Esta ejecución es en orden
  - Primer repetición: corresponde al primer elemento del rango
  - Segunda repetición: corresponde al segundo elemento del rango
  - *n*-ésima repetición: corresponde al *n*-ésimo elemento del rango
  - Última repetición: corresponde al último elemento del rango
- Índice: denota el valor del rango correspondiente
  - Escribiendo su identificador se accede al valor denotado

Rangos de colores

- ¿Qué hace el siguiente procedimiento?
- ¿Cuál es su precondición?

```
1  procedure Arcoiris() {
2    foreach c in [Azul..Verde] {
3      PonerYMover(c, Este)
4    }
5  }
```

Rangos de colores

- ¿Qué hace el siguiente procedimiento?
- ¿Cuál es su precondición?

```
1  procedure ArcoIris() {
2    foreach c in [Azul..Verde] {
3      PonerYMover(c, Este)
4    }
5  }
```

- ¡Los colores también pueden ser índices!
  - Orden alfabético
  - Usar `[minColor()..maxColor()]` para recorrer todos los colores

Rangos de direcciones

- ¿Qué hace el siguiente procedimiento?
- ¿Cuál es su precondición?

```
1  procedure DibujarCuadradadito(color) {
2    foreach dir in [Norte..Oeste] {
3      Mover(dir)
4      Poner(color)
5    }
6  }
```

Rangos de direcciones

- ¿Qué hace el siguiente procedimiento?
- ¿Cuál es su precondición?

```
1  procedure DibujarCuadradadito(color) {
2    foreach dir in [Norte..Oeste] {
3      Mover(dir)
4      Poner(color)
5    }
6  }
```

- ¡Las direcciones también pueden ser índices!
  - Orden en el sentido de las agujas del reloj desde `Norte`
  - Usar `[minDir()..maxDir()]` para recorrer todas las direcciones

Repetición indexada (Resumen)

- La repetición indexada ejecuta *<bloque a repetir>* para cada elemento de un rango [*<primero>*..*<ultimo>*]

Repetición indexada (Resumen)

- La repetición indexada ejecuta *<bloque a repetir>* para cada elemento de un rango [*<primero>*..*<ultimo>*]
- *<primero>* y *<ultimo>* son expresiones

Repetición indexada (Resumen)

- La repetición indexada ejecuta *<bloque a repetir>* para cada elemento de un rango [*<primero>*..*<ultimo>*]
- *<primero>* y *<ultimo>* son expresiones
- El identificador *<indice>* es un expresión que denota el elemento del rango correspondiente a la repetición actual

Repetición indexada (Resumen)

- La repetición indexada ejecuta *<bloque a repetir>* para cada elemento de un rango [*<primero>*..*<ultimo>*]
- *<primero>* y *<ultimo>* son expresiones
- El identificador *<indice>* es un expresión que denota el elemento del rango correspondiente a la repetición actual
- El rango puede ser numérico, de colores, o de direcciones

Repetición indexada (Resumen)

- La repetición indexada ejecuta *<bloque a repetir>* para cada elemento de un rango [*<primero>*..*<ultimo>*]
- *<primero>* y *<ultimo>* son expresiones
- El identificador *<indice>* es un expresión que denota el elemento del rango correspondiente a la repetición actual
- El rango puede ser numérico, de colores, o de direcciones
- El concepto de bloque se extiende para incluir `foreach`

Repetición indexada (Resumen)

- La repetición indexada ejecuta *<bloque a repetir>* para cada elemento de un rango [*<primero>*..*<ultimo>*]
- *<primero>* y *<ultimo>* son expresiones
- El identificador *<indice>* es un expresión que denota el elemento del rango correspondiente a la repetición actual
- El rango puede ser numérico, de colores, o de direcciones
- El concepto de bloque se extiende para incluir `foreach`
- El tema de valores y expresiones lo repasamos la clase que viene